

Thesis

A PREDICTIVE MODEL FOR URBAN VEHICULAR CONGESTION USING DIJKSTRA'S ALGORITHM

Submitted by

Scott Friend

Department of Civil and Environmental Engineering

In partial fulfillment of the requirements

For the degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2008

This work is licensed under the [Creative Commons Attribution 3.0 United States License](http://creativecommons.org/licenses/by/3.0/us/)~~Creative Commons Attribution 3.0 Unported License~~. To view a copy of this license, [visit](#):

<http://creativecommons.org/licenses/by/3.0/us/>

Formatted: apple-style-span, Font: 11 pt, Font color: Black

Formatted: Centered

[Or](#), send a letter to:

Formatted: Centered

Creative Commons

171 Second Street, Suite 300

San Francisco, California, 94105, USA.

COLORADO STATE UNIVERSITY

November 5, 2008

WE HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER OUR SUPERVISION BY SCOTT FRIEND ENTITLED A PREDICTIVE MODEL FOR URBAN VEHICULAR CONGESTION USING DIJKSTRA'S ALGORITHM AS FULLFILLING IN PART REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE.

Committee on Graduate work

Dr. L. Darrell Whitley

Dr. John van de Lindt

Dr. Suren Chen

Advisor Dr. Paul Heyliger

Department Head Dr. Luis Garcia

ABSTRACT OF THESIS

A PREDICTIVE MODEL FOR URBAN VEHICULAR CONGESTION USING DIJKSTRA'S ALGORITHM

Managing vehicular traffic in modern, urban road networks is a difficult task. Few traffic models analyze traffic on a per-car basis over a wide network of roads. This is because of the inherently uncertain nature of traffic engineering. In such a random environment, uncertainty abounds.

Gains in computer processing power have allowed for greater resolution of problems related to traffic engineering. This study analyzes a traffic network with a per-car resolution and three-second time increment. Vehicular pathing implemented Dijkstra's algorithm coupled with normally distributed, random speeds. Additionally, vehicle accidents shut down main arteries at defined intervals. The study evaluated three similar situations, each varying by the number of inactive arteries. The results indicated that the usage of Dijkstra's Algorithm to model traffic is effective and that such a model can successfully traverse blockages and traffic with many thousands of cars.

Scott Friend

Department of Civil and Environmental Engineering

Colorado State University

Fort Collins, CO 80523

Fall 2008

1.	Introduction	1
2.	Literature Review	4
3.	Modeling Concepts	6
3.1.1.	Dijkstra's Algorithm.....	6
3.1.2.	Dijkstra's Algorithm with Normally Distributed Edge Weights.....	9
3.1.3.	Monte Carlo	10
3.2.	Additional Methods	12
3.2.1.	Driver Aggressiveness Modeling	12
3.2.2.	Road avoidance/closure Modeling	13
3.2.3.	Arbitrary Vehicle Start and End Locations	13
3.2.4.	Assigned Start Times	14
3.2.5.	Impossible path logic	14
3.2.6.	Traffic Algorithms.....	15
3.2.7.	Selectable Time Resolution	16
3.2.8.	Per-Car Resolution	17
4.	Procedures and Application.....	18
4.1.	Graph Configuration	18
4.1.1.	Stadium	18
4.2.	Town	19
4.3.	Failure Modes	20
4.3.1.	Horsetooth Road and Drake Road Failure	21
4.3.2.	Horsetooth, Drake, Prospect and Harmony Road Failure	21
4.3.3.	No failures	22
4.4.	Departures and Destinations	22
4.5.	Map	23
4.6.	Initialization.....	24
4.6.1.	Vector Relationship Construction	26
4.7.	Primary Iterative Loop	26
4.7.1.	Case A: Bounding Edge Vector Reached	28
4.7.2.	Case A: Vehicle Has Reached Destination.....	29
4.7.3.	Case B: Vehicle Has Not Reached Destination.....	29

5. Recommendations	3
5.1. Signaling	31
5.2. Acceleration and Stopping Sight Distance	32
5.3. Lane selection and lane switching	32
5.4. Interface with GIS	33
5.5. Code Port	33
5.6. Additional Pathing Algorithms	33
5.7. Left-Hand Turns	34
5.8. One-Way Streets	34
5.9. Lane Orientation	34
5.10. Dynamic Destinations	35
5.11. Improved Normal Random Layer	35
5.12. Epiphany Drivers	36
5.13. Additional Improvements	37
5.14. Collision Detection	37
6. Summary and Conclusions	3
7. Works Cited	4
8. Appendices	4
1. Introduction	
2. Literature Review	
3. Modeling Concepts	
3.1.1. Dijkstra's Algorithm	6
3.1.2. Dijkstra's Algorithm with Normally Distributed Edge Weights	9
3.1.3. Monte Carlo	10
3.2. Additional Methods	12
3.2.1. Driver Aggressiveness Modeling	12
3.2.2. Road avoidance/closure Modeling	13
3.2.3. Arbitrary Vehicle Start and End Locations	13
3.2.4. Assigned Start Times	14
3.2.5. Impossible path logic	14
3.2.6. Traffic Algorithms	15

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted: Default Paragraph Font, Check spelling and grammar

Formatted

3.2.7. Selectable Time Resolution.....	16	Formatted	...
3.2.8. Per-Car Resolution.....	17	Formatted	...
4. Procedures and Application.....	18	Formatted	...
4.1. Graph Configuration.....	18	Formatted	...
4.1.1. Stadium.....	18	Formatted	...
4.2. Town.....	19	Formatted	...
4.3. Failure Modes.....	20	Formatted	...
4.3.1. Horsetooth Road and Drake Road Failure.....	20	Formatted	...
4.3.2. Horsetooth, Drake, Prospect and Harmony Road Failure.....	21	Formatted	...
4.3.3. No failures.....	21	Formatted	...
4.4. Departures and Destinations.....	22	Formatted	...
4.5. Map.....	23	Formatted	...
4.6. Initialization.....	23	Formatted	...
4.6.1. Vector Relationship Construction.....	25	Formatted	...
4.7. Primary Iterative Loop.....	25	Formatted	...
4.7.1. Case A: Bounding Edge Vector Reached.....	27	Formatted	...
4.7.2. Case A: Vehicle Has Reached Destination.....	28	Formatted	...
4.7.3. Case B: Vehicle Has Not Reached Destination.....	28	Formatted	...
5. Recommendations.....	30	Formatted	...
5.1. Signaling.....	30	Formatted	...
5.2. Acceleration and Stopping Sight Distance.....	31	Formatted	...
5.3. Lane selection and lane switching.....	31	Formatted	...
5.4. Interface with GIS.....	32	Formatted	...
5.5. Code Port.....	32	Formatted	...
5.6. Additional Pathing Algorithms.....	32	Formatted	...
5.7. Left-Hand Turns.....	33	Formatted	...
5.8. One-Way Streets.....	33	Formatted	...
5.9. Lane Orientation.....	33	Formatted	...
5.10. Dynamic Destinations.....	33	Formatted	...
5.11. Improved Normal Random Layer.....	34	Formatted	...
5.12. Epiphany Drivers.....	35	Formatted	...

5.13. Additional Improvements	36	Formatted: Default Paragraph Font, Check spelling and grammar
5.14. Collision Detection	36	Formatted: Default Paragraph Font, Check spelling and grammar
6. Summary and Conclusions	3	Formatted: Default Paragraph Font, Check spelling and grammar
Works Cited	4	Formatted: Default Paragraph Font, Check spelling and grammar
Bibliography	4	Formatted: Default Paragraph Font, Check spelling and grammar
10. Appendices	4	Formatted: Default Paragraph Font, Check spelling and grammar
		Formatted: Default Paragraph Font, Check spelling and grammar
		Formatted: Default Paragraph Font, Check spelling and grammar
		Formatted: Default Paragraph Font, Check spelling and grammar
		Formatted: Default Paragraph Font, Check spelling and grammar
		Formatted: Default Paragraph Font, Check spelling and grammar

Table of Figures

Figure 1: Dijkstra's Algorithm Iterative Logic..... 7

Figure 2: Two Monte Carlo Trials..... 11

Figure 3: Traffic reduction coefficient versus overcapacity..... 16

Figure 4: West Fort Collins..... 19

Figure 5: Modeled area..... 20

Figure 6: Initialization Variables..... 25

Figure 7: Mean time to destination 39

Figure 8: Travel time impacts..... 40

Figure 9: Map zones..... 45

Figure 10: Epiphany time as a function of total time 46

Figure 11: Vehicles active vs. total capacity 47

Figure 12: Mean speed over time..... 48

Figure 1: Dijkstra's Algorithm Iterative Logic..... 7

Figure 2: Two Monte Carlo Trials..... 11

Figure 3: Traffic reduction coefficient versus overcapacity..... 16

Figure 4: West Fort Collins..... 19

Figure 5: Initialization Variables..... 24

Figure 7: Map zones..... 42

Figure 8: Epiphany time as a function of total time 43

Figure 9: Vehicles active vs. total capacity 44

Figure 10: Mean speed over time..... 45

Formatted: Font: Calibri, 11 pt, Not Bold, Font color: Auto

Formatted: Normal, Tab stops: Not at 5.99"

Formatted: Line spacing: 1.5 lines

Formatted: Table of Figures, Left, Line spacing: 1.5 lines, Tab stops: 5.99", Right,Leader: ...

1. Introduction

With the advent of powerful and affordable computing platforms, the ability to model traffic flow with significant detail and resolution in near real-time is now possible. Creating a robust traffic model today is easier with modern traffic analysis software. However, nearly all available models rely on existing traffic counts. While providing exceptional utility for traffic growth, signaling and traffic optimization, current traffic analysis software packages are not of great use to model atypical events ranging from the mundane, such as large sporting and musical events, to crises, including natural disasters or military actions.

Traffic flow that is significantly outside expected conditions would benefit greatly from predictive modeling. Consider a football game. Law enforcement manually operates traffic signals in Fort Collins, Colorado, during large events. This is expensive both from a monetary and social perspective; automated signals now incur the added operational costs of manual guidance and the community is forced to commit socially valuable law enforcement resources to manage traffic.

Such predictive modeling would be of great use if a metropolitan center were to unexpectedly empty in a rapid manner. In this situation, the normal signaling would be inadequate. Predictive modeling would allow city authorities to implement a preplanned alternative signaling pattern for the area, allowing for greater flexibility in dealing with the surge in vehicles.

The model used in this study is a predictive traffic package, designed to simulate atypical traffic conditions. It used to examine different scenarios with varying conditions to locate potential bottlenecks in the grid. The results may ~~help formulate~~ help formulate contingency signaling patterns or to aid law enforcement in easing congestion at select locations. The software allows

authorities to maximize scarce resources and shape traffic in a controlled manner with a good understanding of aggregate vehicle behavior.

The traffic model has four primary components: data input and processing, vehicle management, road closure and the pathing algorithm. Data entry into ~~the model~~ the model is via tables describing the properties and geometry of the road network along with the state and preferences of all vehicles. The data is then run through a the model by managing each vehicle's location, situation and current path on the map, with each vehicle evaluating its location and path using the pathing algorithm as fit.

The user enters all data in tabular form. The model requires artery and speed limit flags for each road, with starting and ending points defined. In addition, each street requires its own mean and standard deviation speeds entered along with sectional capacity. The user provides a separate table giving GPS data for each for the start and end of each road. From this, the model automatically calculates the length of each road and creates a table defining road connection relationships.

When the model is run, each car is kept in a vehicle state table recording the vehicle's unique car identification number, starting location, destination, start location of its current road, end location of its current road, distance into current road, distance of road it is traveling on, if the vehicle has reached its destination and the driver's aggressiveness (if selected). These variables update at every time increment for each car as needed. This logic is quick and performs the necessary bookkeeping.

Roads often fail while under extraordinary traffic volume, largely from accidents. Streets may also close due to damage or flooding. The model incorporates road closure logic that allows the

user to shut down any number of streets at any time and duration. This gives the user an ability to force dynamic traffic adjustments as roads close and open. Another use is to force a greater number of vehicles to traverse a certain path and can help identify bottlenecks. This component is optional.

The pathing algorithm calculates each vehicle's desired path. The model allows for the use of any pathing algorithm. The model as used in this study applied Dijkstra's Algorithm with a normal, random layer to determine vehicle pathing. Dijkstra's Algorithm is a shortest-path algorithm that takes as input (in this case) a traffic map, a starting location and an ending location, and returns the quickest path. The normal, random layer computes a constantly changing set of road conditions derived from a normal curve that deviate from the actual conditions of the road. Thus, each vehicle has its own preference when determining desired paths. The normal random layer gives a more realistic approximation of human path finding that varies slightly per-person and over time. The pathing algorithm performs the bulk of computational effort.

The predictive algorithm for congestion is a computational scheme that takes advantage of modern computational processing power to identify traffic patterns preemptively and localized congestion for atypical events and traffic situation. This differs from many current traffic analysis packages that focus on urban planning and which discount emergency or worst-case scenarios. It utilizes Dijkstra's Algorithm to compute paths efficiently and scales down to a per-car resolution for greater accuracy. The model also includes the ability to dynamically shut down and reopen roads, forcing traffic to reroute. These capabilities would help planners preemptively reallocate limited traffic resources.

2. Literature Review

This study incorporates the principles of graph theory and path finding, as well as traffic theory. Dijkstra's Algorithm, published in 1959, details the algorithm (Dijkstra, 1959). Its use is typically confined to computer science problems (Cormen, Lieserson, Rivest, & Clifford, 2001), but also has applications in other domains. Dijkstra's algorithm is a shortest-path function that takes a series of interconnected paths with varying travels times and returns the quickest path in an efficient manner, avoiding an exhaustive search.

Previous studies have examined the use of Dijkstra's Algorithm for traffic modeling. A study examined its utility for single- vehicle routing (Luyao, Zhou, Li, & Chen, 2007). This approach considered traffic as a random car found the most efficient path to its destination. A similar study used a Dijkstra-like method, with the difference in the dynamic modeling of edge costs (Leurent & Aguilera, 2005) . A separate study looked at random effects coupled with Dijkstra's Algorithm in computer science applications (Doumith, Gagnaire, Audouin, & Puech, 2005) where the demand for Dijkstra's Algorithm routing was random but intermediary variables remained constant.

~~Other~~ Shortest-path algorithms have been used to compute paths for vehicular traffic (Huang, Wu, & Zhan, 2007), such as the A* algorithm, which is similar to Dijkstra's Algorithm but attempts to determine the distance from its current location to the destination. Others compared the utility of a wide range of algorithms in the shortest-path class using real road network data rather than random maps (Zhang & Noon, 1998).

There have been few attempts at creating a Dijkstra-based traffic model using multiple vehicles with a normal, random number assigned to each car, causing vehicles to travel at differing

speeds. This layer of random numbers on top of each driver's pathing preferences allows for a more organic and less rigid flow of traffic. This study uses such a model within a real-world 16-square mile road network containing tens of thousands of vehicles.

3. Modeling Concepts

Three alternative approaches modeled car behavior, with one used to represent a real-world scenario:

- The first method used an implementation of Dijkstra's algorithm (DA), a class of shortest-path problems. This ~~applied such an~~ algorithm ~~to find~~ finds the quickest path, given a series of linked *vectors* (nodes) and *edges* (lines connecting two nodes).
- A second method coupled random noise with Dijkstra's Algorithm. Such noise allowed for an imperfect driver, who could choose less-optimal paths. This approach examined the algorithm under a better approximation of real-world conditions.
- A third method ran a Monte Carlo simulation on top of a dynamically generated exhaustive pathing table. The Monte Carlo model generated random numbers for all possible routes, allowing for less rigid pathing than either of the first two methods did. However, the Monte Carlo method would be unsuitable for real-world implementation because of impractically high computational cost.

3.1.1. Dijkstra's Algorithm

Dijkstra's algorithm falls under the broad class of *graph traversal* algorithms in the combinatorics field. Graph traversal algorithms apply to problems ~~where traversing~~ where ~~traversing~~ a given set of connected lines and ~~nodes in~~ nodes in an optimized manner is required.

Over a dozen of these algorithms exist, each adept at solving for specialized conditions in a space (*graph*). This study will apply Dijkstra's Algorithm, which is used for problems where the shortest cost (*travel time*) is sought, contingent on there being a single start *vector* (node), and no *edges* (lines) associated with a negative cost. A cost in the graph search lexicon may be any

penalty incurred while traversing an edge. Examples include time taken, resources used or distance traveled, among other possibilities.

Consider Dijkstra's Algorithm as a series of five iterative steps that return the shortest path for a given graph, as visualized in [Figure 1](#) **Error! Reference source not found.**

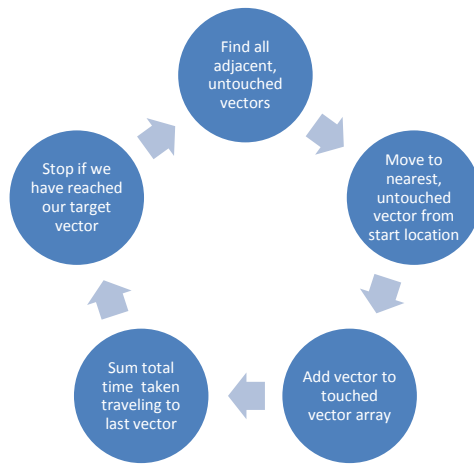


Figure 1: Dijkstra's Algorithm Iterative Logic

The mathematical definition of Dijkstra's Algorithm is as follows (Dijkstra, 1959):

- ❖ An edge weight array is generated so that we may reference edge weights for any the vectors (i,j) with a common edge, a mean edge weight of d_{ij} and a starting vector, s .

$$D_j = \begin{cases} 0 & j = s \\ d_{sj} & j \neq s \text{ and has an edge to } s \\ \infty & j \neq s \text{ and has no edge to } s \end{cases} \quad (3-1)$$

Comment [EU1]: Should these be numbered? I don't know when to start.

Comment [S2]: I'm not sure what you're asking. I assume any figures would be sequentially ordered as they appear in the work. What am I missing?

- ❖ As we may not backtrack, the touched vector array will keep track of all vectors through which we have traveled.

$$\Omega = \{s\} \quad (3-2)$$

- ❖ We search the edge weight array for all open vectors that are not in the touched vector array and connect by a single edge to Ω .

$$i \notin \Omega \ (i, j = 0, 1, 2, \dots, N - 1) \text{ such that } D_i = \min_{j \notin \Omega} D_j \quad (3-3)$$

- ❖ For all $j \notin \Omega$, find the vector with the least edge weight from in ~~(3-3)(3-3)~~

Formatted: Font: Bold

$$D_j \leftarrow \min[D_j, D_i + d_{ij}] \quad (3-4)$$

- ❖ The nearest, untouched vector as found in ~~(3-3)(3-3)~~ is appended to the touched vector array; this prevents any edge from being traveled through more than once. If we append our target vector, stop. Otherwise, go to ~~(3-3)(3-3)~~.

Formatted: Font: Bold

Formatted: Font: Bold

$$\Omega \leftarrow \Omega \cup \{i\} \quad (3-5)$$

Like many algorithms in the graph traversal class, Dijkstra's Algorithm is decisively quicker than an exhaustive-search method. Its iterative, step-based nature requires few logical operations resulting in efficient searching. Yet, the algorithm is also perfect, never failing to find a sub-optimal path. There is no error; the result is always the quickest path for any valid input.

3.1.2. Dijkstra's Algorithm with Normally Distributed Edge Weights

Drivers do not always find the shortest path. Rather, the quickest path is not certain, but rather a probability, contingent on a myriad of inputs. Conditions such as increases in traffic on a given road or unexpectedly fast travel times on a normally slow route are typical. A layer of randomness is desirable in any traffic model to account for the unexpected. The following approach formulated in this study made use of Dijkstra's Algorithm to compute paths, but generates a normally distributed speed.

This second approach derives from the Dijkstra's Algorithm approach above, but with ~~(3-6)(3-6)~~ inserted after step ~~(3-2)(3-2)~~ and prior to ~~(3-3)(3-3)~~, where σ_{ij} is the standard deviation of the mean edge speed, d_{ij} .

$$D_j = \begin{cases} 0 & j = s \\ d_{ij} + \text{randnum} * \sigma_{ij} & j \neq s \text{ and has an edge to } s \\ \infty & j \neq s \text{ and has no edge to } s \end{cases} \quad (3-6)$$

With the inclusion of, ~~(3-6)(3-6)~~, variations in edge weights, or normal random layer, across the graph realize and static edge weights disappear. This results from assigning not only a weight to each edge, but also a standard deviation. A random number generates from every edge's Gaussian distribution of possible weights, making slower routes faster on occasion and opening up a breadth of pathing diversity. This runs at every time increment and the shortest path is recalculated using Dijkstra's Algorithm. The result is drivers constantly reevaluating the perceived shortest path, and deviating from their previously chosen route as they see fit.

Comment [EU3]: Again, need a reference.

Comment [S4]: That is my unique approach. I took Dijkstra's Algorithm and added a simple random element to it. I've tried to clarify that it's my approach, not that of others.

Formatted: Font: Bold

Formatted: Font: Bold

Formatted: Font: Bold

Formatted: Font: Bold

3.1.3. Monte Carlo

The Monte Carlo method is a broadly defined class of stochastic computational algorithms. A stochastic approach proves its worth when seeking every combination of an algorithm, but there are inadequate resources or time to complete the task. A Monte Carlo approach will try a random subset of all algorithmic combinations and return an approximate solution based on these incomplete, but sufficiently diverse, *trials* (iterations).

This Monte Carlo method may separate into four distinct stages. It then creates a list of all possible paths before starting the Monte Carlo procedure. Each trial generates, a normal, random number for each edge in every path starting and ending at the desired vectors. This method then sums each complete path's cost with the normally adjusted edge weights. The method then calculates the least-costly paths from each trial and a selects a final path based on a weighted probability. The steps below enumerate this approach:

- ❖ The first course of action is to generate an exhaustive path list. Two methods are available for this: brute force or traveling each path. Traveling each path is a more efficient solution compared to brute-force searching, as the vast majority of potential paths do not actually exist. This is similar to Dijkstra's Algorithm. Assigned to each traveler is a starting and ending vector, with every path followed to completion. If the end vector is the desired destination vector, exhaustive path list appends the current path. Otherwise, the method discards the path. Calculation of the exhaustive path list only executes once, providing easy referencing thereafter. This method is not practical in reality, due to a need for an exhaustive path list. The amount of time needed to find all paths grows exponentially and would only be of use for small problems.

- ❖ A normal, random number adjusts each edge weight of each path. For example, edge n receives a different random number for each path in which it is present. Thus, for two paths {A-B-C} and {A-D-E}, A will be assigned a differing normal, random number for each path. This is repeated for each trial; the random numbers generated each edge in {A-B-C} in trial one would be different for the same edges in trial 2.

Consider [Figure 2—Error! Reference source not found.](#), which contains a simple graph with three edges. This graph, if run through two Monte Carlo trials, would result in a path tree as seen in the bottom of the [figure](#).

Decision Tree: Two Monte Carlo Trials

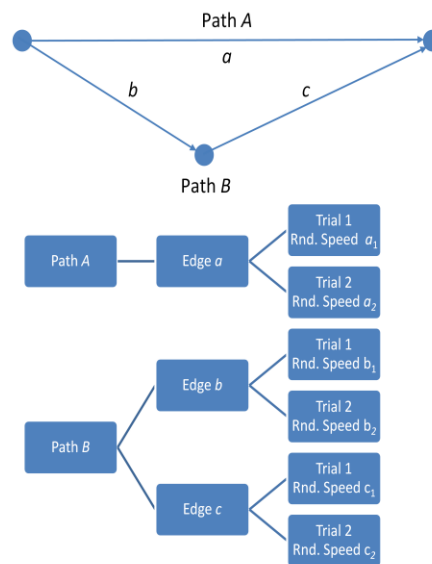


Figure 2: Two Monte Carlo Trials

- ❖ The total cost of each path, for each trial is calculated. Consider [Figure 2](#) again—[Error! Reference source not found.](#) Path B consists of two edges, thus the total cost for trial 1 would be $a_1 + b_1$ and for trial 2, $a_2 + b_2$. Never matched are values from differing trials.
- ❖ It selects the path with the minimum cost from each trial, resulting in a final selection between n paths for n trials. These paths then receive a weight based on their total cost. For

Comment [EU5]: Are you going to put your figures on the same page where they are initially referenced? That would be extremely helpful.

Comment [S6]: All figures that elegantly fit I put in. The triple-figures (which I found no clever way to condense into one single figure) I left in the appendix. Should it be best to integrate those then let me know and I'll devise something.

example, take two paths A and B with respective total costs of five and one. Assigning each path a weight based on their total cost would, for A return a weight of $1 - \frac{5}{6} = 0.1667$, and $1 - \frac{1}{6} = 0.8333$ for B . A randomly generated, uniformly distributed number between zero and one selects a weight. B is chosen if $0 \leq B \leq 0.8333$ and A chosen when $0.8333 < A \leq 1$.

3.2. Additional Methods

3.2.1. Driver Aggressiveness Modeling

Not all drivers show the same driving habits. While some will weave between cars, others wish to drive below the speed limit. Lacking the resources to model driving habits in-depth, an approximate model can be made by assigning a driver aggressiveness coefficient. The model assigns this coefficient a mean of zero and a user-defined standard deviation that may vary. However, an empirically derived value of 0.1μ mi/hr gave the best results. For a road with a speed limit of 30 mph and aggressiveness coefficient of 0.1μ mi/hr:

- Sixty-eight percent of drivers would travel within 3 mph of 30 mph.
- Ninety-five percent of drivers would travel within 6 mph of 30 mph.
- Over 99% of drivers would travel within 9 mph of 30 mph.

The model chooses a random number from a Gaussian distribution for each car, and multiplies it against each edge's weight. Each vehicle modeled keeps this coefficient for the entirety of the simulation.

Comment [S7]: I'm not sure I follow. That the simulation's properties aren't tangible (by definition) should be clear to the reader. Or is this not the case? If my minor changes are not sufficient please let me know.

3.2.2. Road avoidance/closure Modeling

Roadwork is a complicating factor when traveling and road closures are an integral part of any driver's mapping calculus. In such cases, they must alter their travel around closures, resulting in the diversion of traffic off on other, less capacious, roads. Under other circumstances, roads may close because of accidents, natural phenomena such as flooding and debris or events at a nearby venue.

The model treats these events in a binary fashion; an edge will either accept or deny vehicles. It assumes that each vehicle will have knowledge of road closures as they happen, presumably from radio, cell phones or other media. Furthermore, as the accident clears or debris swept away, roads will resume normal operations. In this simulation, a corresponding list of "failure" and "revive" times are kept, allowing edges to be dynamically switched on and off by the simulation as the user sees fit.

A list of closed roads, R , containing the bounding vectors $\{i,j\}$ of the closed edge and its corresponding failure-reopen $\{s,t\}$ times is kept by

$$R \ni \{i, j, s, t\} \quad (3-7)$$

Each chosen path validates against R . The algorithm recalculates the route if it travels through a closed edge. This executes until it finds the best, legitimate course.

3.2.3. Arbitrary Vehicle Start and End Locations

A critical part of any large traffic model is an arbitrary set of starting and ending locations for vehicles, with such diversity found by placing each vehicle as a row in an array, V_c , with corresponding starting, s , and ending, t , locations that the pathing algorithm may reference.

$$V_c = \{s, t\} \quad (3-8)$$

This list may be useful for automating different traffic patterns by generating two columns of random integers from one to the end vector index, assuming that all paths are reachable.

3.2.4. Assigned Start Times

In order to facilitate a more gradual loading of the modeled road grid, the model allows each car to receive a specific starting time. Situations requiring such a requirement arise when considering traffic flow after the conclusion of large events, where the venue has the capacity to release only a specific number of vehicles per unit time. Of special import are sporting, cultural, musical and political events, where the largest and best-managed city grids may only accommodate a fraction of the vehicular deluge.

The present model accounts for this behavior by containing simple scheduling logic. This logic references a user-provided list of release times for each car, r_c . Such releases are independent of starting location, ending location or any other input factors and checked only at the start of each time increment.

$$V_{c,3} = r_c \quad (3-9)$$

3.2.5. Impossible path logic

In certain rare conditions, a car may find no possible pathing solution. This is not the fault of Dijkstra's Algorithm or Monte Carlo logic, but arises when too many paths are closed or if the traffic algorithm overloads an edge. In this case, there are no pathing options for the vehicle.

Under these conditions, the model implements simple logic to halt all progress on car travel along the dead edge, and check conditions at every time interval until a possible path emerges. So long as the destination remains out of reach, the stranded vehicle will remain idle. This is of great use when a vehicle is trapped in a dead edge and unable to exit off a side street, modeling congestion.

3.2.6. Traffic Algorithms

Traffic and congestion are not binary events. Rather, they correlate with overcapacity on a per-road basis. It is difficult to model the correlation between road capacity, current vehicle loading and mean road speed. In the interest of flexibility, the model allows for any desired traffic speed reduction algorithm, which may, but is not required, to be used as seen fit. Such a *speed-reduction coefficient* schema allows for any value resulting in a coefficient of $0 \leq f(t) \leq 1$, with a value of one representative of uninhibited traffic flow and a linear speed decrease as the coefficient decreases. All traffic on the edge will halt when the speed reduction coefficient falls to zero.

The traffic function, $f(t)$, used for the simulation with car capacity, c , and number of cars currently loaded, n , (Abramowitz & Stegun, 1972)

$$f(t) = \begin{cases} n \leq c & 1 \\ n > c & 1 - \int_{-\infty}^{\infty} \frac{1}{\frac{c}{5}\sqrt{2\pi}} e^{\frac{-(x - (1.25c + \frac{c}{5}))^2}{(\frac{2c}{5})^2}} dx \end{cases} \quad (3-10)$$

is an integral similar a cumulative density function. The result is an accelerating speed reduction as usage increases, with a decrease in the rate of change in speed reduction as loading passes 125% capacity. Past this volume, further vehicle load will result in smaller reductions in mean travel speeds. The quality of both the traffic model and speed reduction coefficient profoundly affects the results of the model. ~~Figure 6~~Figure 5 presents ~~the visual~~the visual relationship between speed reduction coefficients versus road capacity, as described by ~~(3-10)~~(3-10).

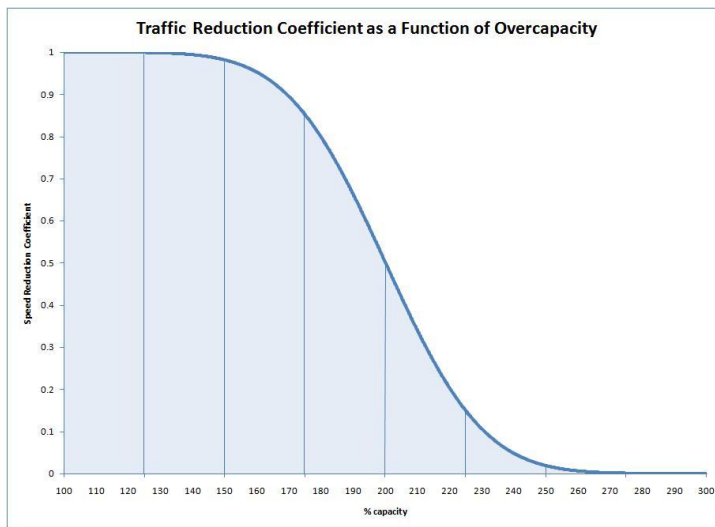


Figure 3: Traffic reduction coefficient versus overcapacity

3.2.7. Selectable Time Resolution

The time resolution may be set to allow differing degrees of accuracy. The length of each time slice is important, and best minimized to reduce rounding errors. The unit of time used is minutes, with any integral, or fraction thereof, acceptable. A resolution of 0.1 minutes or less is favored, computational brevity aside. If the resolution is too large, cars will travel at incorrect speeds when entering edges. This error arises from the extra distance traveled beyond the previous edge during the time increment.

For example, if a vehicle has 0.01 miles to travel on the current edge, but during the following time increment is able to travel 0.03 miles, it will overshoot the edge. Any remaining distance it may travel, in this case 0.02 miles carries to the next edge it traverses during that same time increment. The next edge may have a different average speed or traffic conditions, but the model would disregard any environmental variables and the vehicle would travel 0.02 miles irrespective of actual road conditions. Time restraints did not allow for a fix.

3.2.8. Per-Car Resolution

While the computational cost of determining the possible path choices for each car is significant, this method provides a more accurate and higher resolution for modeling than a macro, clustered-car approach. Each car acts independently, computes or receives:

- A driver aggressiveness coefficient
- Start and end vectors
- Calculates the preferred graph traversal algorithm, be it Dijkstra's Algorithm or Monte Carlo
- Separate plotting
- Calculation of upcoming traffic conditions
- Exact location on the graph
- The assigned start time
- Unreachable target logic

4. Procedures and Application

4.1. Graph Configuration

The model used the town of Fort Collins to examine its utility. ~~To mobilize~~To mobilize a large number of cars on the road, it assumed that there was a rapid and precipitous loss of public confidence in the integrity of a nearby dam situated to the west. Such an event would cause the majority of the local population to flee due east within a short time window. The simulation placed over thirty thousand vehicles around town, as well as a sports stadium filled to capacity. The latter generated overwhelming congestion radiating from a singular point that tested the model's ability to reroute epiphany drivers (Section 5.12, p. ~~3635~~) around traffic as conditions dictated.

4.1.1. Stadium

Fort Collins is home to Hughes Stadium, a collegiate sporting venue with a seating capacity of 34,000, and is located at the west end of town as shown in ~~Figure 4. Error! Reference source not found.~~ The dam lies nearby to the west. Should any fear arise over the dam's ability to operate safely, one assumes the stadium would rapidly empty.

The stadium contains three exits, with two lanes per exit. When operated at maximum flow, empirical observations suggest 1.67 vehicles per lane per second may exit at peak capacity. This maximum exit rate is only valid if the streets upon which the cars exit are not experiencing any traffic burden.

For every two spectators present one vehicle departed. With the stadium filled to maximum capacity, this loosed 17,000 vehicles attempting to flee as quickly as the flow of traffic allowed.

This will create an extraordinary level of congestion on the roads immediately adjacent to the stadium, and will propagate outward until well after all cars have departed. With no traffic, the departure would complete in approximately 18 minutes. However, all adjacent roads would be unable to handle the crush of cars and the resulting overburden expected to result in significant

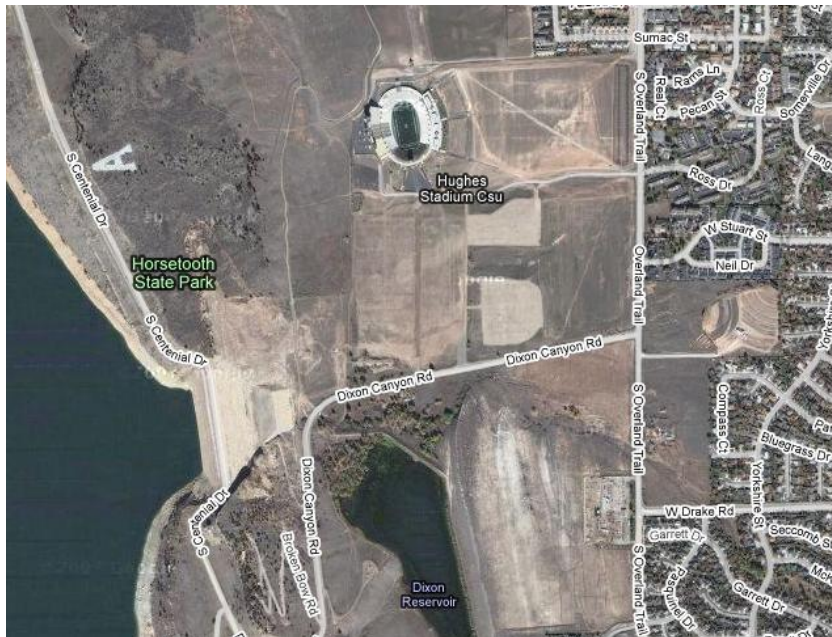


Figure 4: West Fort Collins

delays.

4.2. Town

As of 2006, the Census Bureau estimated Fort Collins' population to be 135,481 residents of all ages residing in 55,784 households ~~(U.S. Census Bureau)~~ (U.S. Census Bureau, 2006) of both

related and unrelated persons. From these residences, 1,827 had no vehicles, while the majority had two. However, not all residents are located on the graph used for the model. The simulation assumed that 40,000 residences fell within the graph's bounds of 16 square miles and that 95% of the households departed with one car, putting 38,000 vehicles



Figure 56: Modeled area

on the road. ~~Figure 5~~ Figure 6 shows a satellite image of the area modeled, offering a visual guide to the region's density. Each vehicle left at a random time within a 20-minute interval from the start of the simulation.

Comment [S8]: FIX - But how? What a mess.

4.3. Failure Modes

Accidents generally increase with higher vehicle usage rates. To model this, the affected road disables itself from use for 60 minutes. All such failures will occur among main arteries at

varying intervals and will force vehicles to reconsider their pathing decisions. In addition, any failure will force cars on to less desirable routes, creating considerably more congestion in certain areas than would be expected under similar conditions. Three modes are considered:

- Horsetooth Road and Drake Road failing from Shields Street to South College Avenue
- Horsetooth Road, Drake Road, West Prospect Road and West Harmony Road failing from South Shields Street to South College Ave
- No failures occur.

All failures start 10 minutes after the simulation began, with 1-mile east-west sections failing every five minutes.

4.3.1. Horsetooth Road and Drake Road Failure

Traffic flows eastward, putting considerable strain on all east-west arteries, Horsetooth Road and Drake Road among them. These arteries fail at 10 and 15 minutes, respectively, and ~~remain stay shut~~ down for one hour. During this time, cars are able to cross the roads heading north or south, but otherwise unable to traverse the failed roads. Vehicles traveling on the roads when they failed had to either exit along the nearest side street or idle until the accident cleared and the road reopened.

4.3.2. Horsetooth, Drake, Prospect and Harmony Road Failure

All four of the roads under this failure condition were major east-west arteries and resulted in substantial congestion on the remaining two arteries on the map. Failure occurs at 10, 15, 20 and 25 minutes for Horsetooth Road, Drake Road, Prospect Road and Harmony Road,

respectively. The two remaining arteries, Mulberry Street and Laurel Street, both on the northernmost section of the map, would be a bottleneck with sufficient traffic. All failed roads reopened within an hour after initial failure, but that would likely be of little help to the travelers consumed by gridlock to the north.

4.3.3. No failures

No failure would be the best outcome under the circumstances, but the effects would still be overwhelming congestion. All arteries would fill ~~to well~~ to well over capacity. As the vehicles from Hughes Stadium propagate eastward, the situation would deteriorate further. Yet with all lanes open, vehicles would have substantial choice as to which path they take, resulting in better and more level distribution throughout the graph.

Each vehicle sees the map differently and each car's opinion will vary over time. Therefore, a reasonable observer would expect some vehicles to take suboptimal paths across roads that appear to be a good idea according to their limited information. Due to variations in traffic, these observations may or may not have resulted in a quicker overall route.

4.4. Departures and Destinations

The volume of cars departing was determined by dividing the map into 18 zones, with all but two zones surrounded by an artery. The two unbounded zones were on the periphery of the map, forcing departing vehicles to travel smaller avenues and collector streets before arriving at high-capacity arterial road corridors. Table 2 ~~Error! Reference source not found.~~ shows the number and percentage of departing cars by zone. Zones with a higher density received a higher

percentage of departing cars while sparse or small zones received less. Most cars fled east, exiting one of five arterial exits on the map.

Some vehicles will not flee. These roamers represent drivers unconcerned or undeterred with the panic of the day or driving to assist other households. Four percent of the departing cars placed in this group. These roamers received a randomly selected departing location and destination. They left at within a 20-minute period, similar to the fleeing vehicles.

Destinations for non-roamers depended on their proximity to the nearest exit. Consider Zone ~~Four~~4 as an example. It is one of the easternmost zones, adjacent to two exit paths. As such, all vehicles departing Zone ~~Four~~4 chose to exit via Mulberry Street or Prospect Street. Zones on the west side of the map had more potential exits. Zone Eight's location at the far west of the map left cars without any nearby arteries. Therefore, vehicles departing Zone ~~Eight~~8 took chose several arterial exits. Some Zone ~~Eight~~8 vehicles chose destinations that involved additional traversal, but possibly considered less costly in terms of time. Most chose paths that led to directly to eastbound arteries. [Table 3Error! Reference source not found.](#) shows the relationship between exit artery and vehicle volume, with arteries arranged from northernmost to southernmost.

4.5.Map

A line-and-node system modeled the city's streets. The input considered each interchange and each variation in road geometry, such as a curve, a vector. These then filled in with connecting lines, representing roads, at all vectors. Some line elements were 50 ft. long, others over a quarter-mile in length. Areas where there were pronounced curves, such as the large one in

Zone 10, were assigned over five vectors to approximate curvature. In total, there were 3,325 vectors and 4,021 edges traversing 246 linear miles.

Comment [S9]: That's greater than I expected.

Spatial coordinates of each vector were acquired by way of Google Maps, individually selected with geo-coordinates (longitude and latitude) returned. From this ~~data~~data, ~~road~~edge ~~_edge~~ lengths were calculated. Edges received bounding vectors, with map placement based on their two bounding vector coordinates.

Speed limit data was necessary to estimate road speeds and congestion penalties. The City of Fort Collins Traffic Operations provided collector/artery locations, speed limits and lane counts throughout the city, each placed on an edge-by-edge basis. Each edge, in addition to length and vector information, received a speed limit, lane count, capacity value and an artery or collector type. These data provided the raw numbers necessary for deriving congestion and vehicle speed values.

4.6. Initialization

As input, the model took three fundamental pieces of information, as shown in [Figure 6](#)~~Figure 5~~.

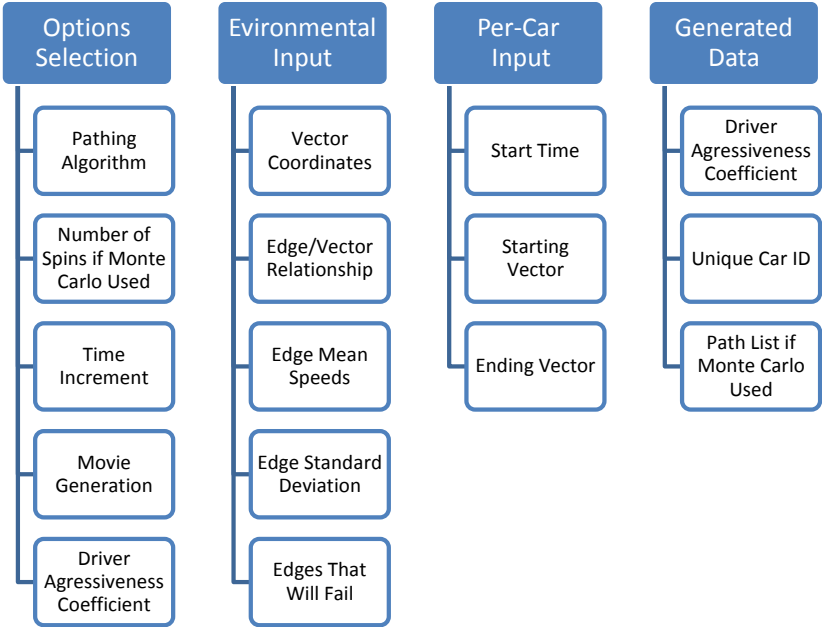


Figure 65: Initialization Variables

Prior to running the model, a number of performance and initialization routines executed to baseline computational costs. The total run time for the model with four failed roads was estimated to be 2 days on a 3.15 GHz Intel Core 2 (Conroe core) CPU with access to 4 GB of DDR2 memory at 735 MHz. The simulation necessitated switching one of the two cores off to allow for the greater stability. The two simulations with two and zero failed roads were run simultaneously on a 2.5 GHz Intel Core 2 (Dothan core) with 4 GB DDR memory at 533 MHz, where each simulation was assigned affinity to one of the two CPU cores. One could expect run time on the latter machine to take longer due to the sharing of the L2 cache and a lower clocking of a practically equivalent core. MATLAB as of the latest release (2008a) does not take

advantage of multi-core CPUs for the routines involved in this model. MATLAB's Distributed Computing Toolbox was unable to reduce computational time, with the toolbox not designed for rapid, repetitive looping. Use of the toolbox considerably slowed down execution speed. The heavily vectorized code offered no performance gains (and in some cases slowed down execution) when ported to MATLAB Compiler.

4.6.1. Vector Relationship Construction

The model placed all edges in a table listing only their bounding vectors. However, a global table showing all edges connected to all vectors was required for quick reference. It then ~~nm~~ built a routine where each row number represented the current vector, and each following column listed any connecting vectors. For example, were vector 1 connected to vectors 2, 5 and 8, the vector relationship table would list row 4 as [2, 5, 8]. This approach allowed for rapid referencing of vector relationships with minimal initial user input and less CPU load.

4.7. Primary Iterative Loop

The Primary Iterative Loop (PIL) performs most of the simple housekeeping tasks without performing time-intensive operations. It runs until all cars have reached their destination. This can cause infinite loops under rare conditions from careless user input. For example, an edge may shut down (to simulate a car accident, for example) and be instructed never to resume accepting traffic. This seems to be a valid condition, but it traps vehicles in the edge when it is shut down, and if they are unable to exit along a side street are be trapped in perpetuity waiting for the edge to revive.

At the top of the loop is the PIL's bad-edge logic. The bad-edge logic checks first to see if any new edges have failed. This is done by evaluating every edge and determining if its fail time is after the current time and before the revive time. Any failed edges (be they newly failed or otherwise) are then copied to a bad edge table.

Memory demands for the simulation are considerable. Consider that each edge has 11 unique attributes and that, for the current model, there are over 4,000 edges. Furthermore, MATLAB requires contiguous blocks of memory when copying matrices or variables. The result is that removing a row from a table forces MATLAB to recopy the entire table to memory. While such an operation is quick on modern computers, when executed tens of millions of times uses considerable resources and requires non-trivial amounts of time to complete. In addition, repeated manipulation of matrices and variables in memory leads to memory fragmentation. As MATLAB allocates memory in contiguous blocks, excessive table allocation in memory increases the likelihood of MATLAB sufficiently fragmenting available memory and resorting to virtual memory. Such scenarios may leads to an order of magnitude in computation time.

To avoid needless table paging, the bad-edge logic will only recreate new edge tables when the number of current bad edges changes. This recreates a complete edge table and removing any failed edges. Any revived edges will be included in the newly generated table since they are already part of the complete edge table.

The PIL then loops through each car in a sub-loop, the Secondary Iterative Loop (SIL). The SIL's first task is to determine if the car either reached its destination or its start time is after the current time, (i.e. it is not ready for release). If either is true, the SIL continues to the next car, requiring no further computation. However, if the car has not reached its destination and

Comment [EU10]: You are making a fine case to ditch matlab and use fortran.

Comment [S11]: Sans punch cards, I would hope.

already been released from its starting location, an additional series of steps are undertaken, dependent on the car's location and pathing method.

A check then determines if the car has reached (or surpassed) the bounding vector on its current edge or if the vehicle is just starting out.

4.7.1. Case A: Bounding Edge Vector Reached

When the car reaches the bounding edge vector, it first subtracts the distance into its current edge from the total length of that edge. For example, if an edge is one mile and the vehicle has traveled 1.01 miles over a specified time increment, the distance into its new edge (that is yet to be determined) is 0.01 miles. This is a source of error, for it assumes that the next edge the vehicle travels on will have similar traffic and speed conditions.

Traffic conditions must discount the car as it leaves its edge. When a vehicle exits an edge, the vehicle count ~~is reduced~~ is reduced by one and congestion recalculated. If the car is just starting out, there is no decrement; rather the vehicle's destination is set to its current point. ~~This~~ This aids Dijkstra's Algorithm in computing the vehicle's starting vector.

The model then adds the vector the car is ~~leaving to~~ leaving to the previous vector array. This step is necessary to prevent the vehicle from backtracking on itself. Such a condition would result in erratic pathing as traffic conditions fluctuated, possibly initiating an infinite loop.

Next, a logical statement evaluates to determine if the car has reached its destination.

4.7.2. Case A: Vehicle Has Reached Destination

If the car has reached its goal (i.e. the edge's bounding vector is the vehicle's target vector), no additional pathing computations need to run again on the vehicle. On a 4,000-edge map, each pathing solution takes up to a half-second or more to compute on a modern computer. With over 50,000 cars, any unnecessary pathing calculation consumes a significant and non-trivial amount of time. To this end, each vehicle row in the vehicle table contains a path-complete flag that identifies vehicles as either path-complete or path-incomplete. Since the vehicle has reached its destination, it is marked as path-complete. It then appends the vehicle's previous car vector with the target vector as well. This is useful when analyzing data after the model completes and is of no use while the model is running. With the car marked as path-complete, the SIL continues to the next vehicle.

4.7.3. Case B: Vehicle Has Not Reached Destination

Since the vehicle has not reached its destination, it reevaluates its own status and recalculates a path to its target. The vehicle takes as input its position on the graph, its destination, any dead edges and current traffic conditions. It then determines if there are any better paths available. However, with the normal edge package enabled, the vehicle would not see the true status of all edges, but a normal, randomly generated value of each edge (the normal, random layer). Thus, the perceived the best path is not necessarily the case. For the current model, the standard deviation was set to 10% of the edge's mean speed. This resulted in paths that were often optimal with reasonable suboptimal pathing deviations. The pathing method selection by the user prior to run-time executes, returning a series of vectors that represents the vehicle's new

path. The vector series starts at the vehicle's current vector and ends with the car's destination vector.

Occasionally no path exists that ends at the vehicle's target vector. In such a case, the pathing algorithm returns a path of zero. This often occurs when a car traps itself in a dead edge because of an accident or the like and is unable to exit off a side street. A logical test checks for this condition and, if found, the SIL continues on to the next vehicle without executing the remaining code. The vehicle remains where it is without moving and reevaluates its options at the next time increment

Its previous current-edge end vector replaces the vehicle's current-edge start vector, while the target destination resets. The program pulls length the vehicle must travel on the current edge from the edge table, and replaces the previous edge's length while the current edge's carload is incremented by one. Then model then calculates the traffic coefficient using the method described earlier, though any algorithm may be used.

The separate conditions of the path complete and incomplete logic converge to update the vehicle's speed. This is necessary whether or not the car has reached the edge's bounding vector, avoided only if the car has just reached its target vector or if there is no possible path to the target and the vehicle is idle.

Having updated the vehicle's position along the edge, the SIL continues to the next vehicle. When it reaches the last row of the vehicle table, it terminates and passes control to the PIL. Having examined all cars, the time is incremented by the user-defined step size. The step size was 0.05 minutes (3 seconds) for this model. Smaller step sizes will result in results that are

more accurate, with an increment of less than 10 seconds strongly recommended based on several runs with this model.

The SIL then performs housekeeping as defined by the user prior to run-time. If the user requests a movie of the resulting flow pattern, then the necessary frame gathering routine executes. The model uses The Windows Media Video 9 Encoder, though any encoder will do. The user can also request data dumps to analyze later. If enabled, all car data since the last dump save to the disk every 100 frames. Writing to disk every 100 frames avoids voluminous memory usage, which otherwise cripples the model.

The PIL prints relevant data to the console as a final step, including the current frame, number of cars path-complete, time taken to process the current frame, total time taken and a time stamp.

A movie is generated (if requested), when the simulation completes. Logging then terminates and control returns to the user.

5. Recommendations

A number of improvements ~~may increase~~ may increase the usefulness of the model. These range from small vehicle logic to substantial signaling implementations

5.1. Signaling

The most pressing improvement would be a signaling model that could accurately manage traffic lights in a fashion similar to the town or municipality under review. Signaling tables would be required, as would a method to stop and start cars as the dictated by the signaling. An additional prediction layer that gives each car-limited insight into signaling patterns would be of additional utility. Currently, all vehicles respect only traffic or dead edges, and flow through intersections without penalty.

5.2. Acceleration and Stopping Sight Distance

This model did not consider vehicle acceleration or deceleration. This significantly affects overall travel times. To accommodate this, speed limits fell by 10 miles per hour. However, this is a crude approximation of reality. Areas of heavy congestion can greatly complicate acceleration and deceleration of traffic, both on a macro and per-car scale. Deceleration models of vehicles are readily available (Transportation Research Board of the National Academies, 2000).

5.3. Lane selection and lane switching

The model currently implements lanes in a limited fashion. It considers only the total number of lanes and returns a capacity value. A more suitable approach would be to model each lane separately. Corresponding vehicle logic would give cars lane selection capability and lane switching capability. Such detailed lane modeling would give the benefit of approaching more organic vehicle modeling on the macro scale. Coupled with an appropriate stopping-sight distance implementation, accurate and detailed models on the street scale would work well.

5.4. Interface with GIS

Building a road table is very time consuming. The 16 square mile section used in this study took approximately 80 hours to build. A more elegant approach would be an option to import GIS data from city or municipal databases, realizing significant time and cost savings.

5.5. Code Port

The development environment for the model is MATLAB. While allowing for rapid development, efficiency suffers and unit testing is unavailable, making debugging difficult. MATLAB's C++ compiler is of limited utility and provides no recognizable gains in performance (and in some scenarios results in slower execution). Furthermore, the utility of object-oriented language would be readily apparent for this model. MATLAB is also not multithreaded, and its built-in tools (as of 2008a) are unwieldy and unsuitable for multithreading scenarios with rapid, but repeated loops. Thus, this model makes no appreciable use of multi-core systems and potential performance gains are unrealized.

If a complete port would be impractical then MATLAB's MEX tools are ~~useful—compiling~~ useful compiling selected code to C++ or FORTRAN, but any speed improvements may be minimal.

5.6. Additional Pathing Algorithms

Dijkstra's Algorithm as implemented worked well. However, the model is well suited for other applications. It requires almost no additional configuration provided additional modeling code is at hand. Only one line of code need change to accommodate any pathing model that accepts the inputs as previously described and outputs a pathing table.

5.7. Left-Hand Turns

There is no time penalty deduction for left-hand turns under the model used in this study. In reality, left-hand turns profoundly affect any vehicle as it travels to its destination. This would pair well with a signaling method as described above.

5.8. One-Way Streets

The model as used in this study is not directionally aware, and cannot evaluate one-way streets.

However, ~~it could be rapidly implemented~~implementation is simple and would be of great utility in downtown or dense urban scenarios, where a great many if not the majority of streets are one-way.

5.9. Lane Orientation

The model currently discounts any lane directionality. That is, cars will drive on the wrong way of the street. This would be small, but not trivial to implement. This would be well suited to for development alongside a lane-selection system. This study removed half the lanes from each road, to a minimum of one lane.

5.10. Dynamic Destinations

Clearly defined definitions are not representative of a real driver. The model currently allows only for a single, unwavering destination. Nevertheless, in many cases it would be best either to have a weighted number of destinations, or fuzzy destinations.

A weighted number of destinations would be a simple implementation method and still allow for great flexibility for the driver to pick their own destination. The user would select two or more destinations. Each weight would be determined either dynamically by the model or by the user. A dynamic weighting selection at the beginning, requiring no user input and with each destination chosen in relationship to the driver's starting point would be of use. Thus, the driver would be inclined to travel to the closest destination (either by direct distance or quickest traveled path), but may choose to travel to a farther destination less often. Alternatively, the user may hard-set the weights, if specific destinations are desired, such as differing onramps to a freeway.

Alternatively, the user could select a fuzzy destination. For instance, all the information the user may have is that most drivers will park "around" a stadium. The model would take that information and, using fuzzy logic, determine a destination choice while it travels. This implementation would be of great use under limited scenarios. Employing this method requires a minimal of fuzzy logic programming with MATLAB using the Fuzzy Logic Toolbox.

5.11. Improved Normal Random Layer

The normal random layer simulates driver uncertainty and changing preferences, but does so inconsistently. Drivers do not change their minds in an evolving fashion. That is, each driver

randomly changes their preferences at defined intervals with no regard for their previous preferences. A more elegant solution would be for the driver to incorporate their previous decisions when determining their current road preferences. An excellent solution would be to incorporate an exponential moving average (EMA) into the driver's consideration logic. An EMA gives considerable weight to recent decisions and far less weight to older decisions, modeled by an exponential curve. The downside to this approach is that an EMA requires a number of prior data points, with 14 suggested, before it can be used (Hunter, 1986). For very large maps EMA logic would be of considerable use, but contribute add little ~~value to~~ value to medium and small graphs.

As an alternative to incorporating EMA logic, a simple prior decision deviation limit would prove useful. This would only allow for a preference that differs by no more than a predefined amount. This would be simple to operate and prevent the driver from wildly changing their minds throughout the simulation.

5.12. Epiphany Drivers

Dijkstra's algorithm, though including a normal, random layer, still proved unrealistically clairvoyant. Cars would usually find the quickest path, with highly correlated pathing choices. This resulted in wide swaths of the map left untouched while a surge of cars backed up major arteries. Each driver having no knowledge of upcoming traffic conditions and seeing taking the same ideal path as their peers caused this. The driver's ideal path would often differ markedly from the actual, congested map. The introduction of the "epiphany driver" concept mitigates this effect.

While typical drivers have no knowledge of upcoming road conditions, epiphany drivers have access to all road conditions at the time when calculating pathing decisions. Thus, an epiphany driver can take timesaving shortcuts though neighborhoods travel longer but less-congested routes and perform limited load balancing. This approximates most drivers taking a familiar path with moderate consistency, with a minority of travelers taking shortcuts and cutting through neighborhoods.

Drivers had a 0.5% chance of becoming an epiphany driver at every time step, unless they already were such a recipient. Epiphanies lasted two minutes, after which the driver would lose situational awareness of traffic. As drivers lost epiphany status, they became eligible again to become an epiphany driver.

5.13. Additional Improvements

The code would additionally benefit from the implementation of more specific algorithms, such as those for parallel parks and stoplight or left-hand turn queuing. Right hand turn models would be of additional use, but could be complex to implement with a definite, but limited, utility.

5.14. Collision Detection

Vehicles currently travel without disregard for the space they are occupying. Any number of cars may share the same physical space. A more appropriate implementation would force each car to respect its peers, with a given distance separating vehicles from one another. This would require a substantial time commitment to implement.

6. Summary and Conclusions

A brief summary of each simulation's results is found in [Error! Reference source not found.](#) Table 1 displays the results of each simulation, with total time defined as the time the simulation took from the start of the evacuation until the last vehicle exited the map. In some

cases, such as Simulation 1, the results are misleading, for all cars but one finished well before the listed total time. That one car was responsible for a non-trivial

Sim.	Scenario	Cars	Total time (min.)
1	No failed arteries	51,600	124.90
2	Two failed arteries	51,600	128.80
3	Four failed arteries	51,600	129.30

Table 1: Simulation results

increase in the listed times from

[Error! Reference source not found.](#) Table 1.

Congestion was considerable as the number of vehicles increased. Cars entered the grid at a uniformly random time between zero and 20 minutes. This manifested itself in the build-up of vehicles followed by a rapid initial decline as seen in [Figure 11](#)[Error! Reference source not found.](#)[Error! Reference source not found.](#). The rate at which vehicles drain softens with time, and depending on the scenario examined, will flux over time.

Yet, when examined in aggregate, mean speeds did not fall dramatically. Small subsets of the main arteries quickly degraded, while the remaining roads on the grid functioned within capacity. [Figure 12](#)[Error! Reference source not found.](#) shows the normalized mean speed at each time increment for all three simulations. The simulations transpire, as one would expect,

with the worst-case situation reporting the least desirable mean speeds. Once all failed arteries resume operation the mean speeds fall roughly in line with each other. However, Simulation 1 enjoys markedly better mean speeds when arteries start to fail.

A chart of each driver's epiphany time as a percentage of total driving time, ~~seen -as seen in~~ Figure 10, ~~by~~ Error! Reference source not found. confirmed that the approach sped up travel times. One can see that the less epiphany time each driver receives, the longer it takes him or her, on average, to reach the destination. The effect was not drastic, but effective in distributing load. This did have the effect of causing considerable congestion on side streets as epiphany drivers elected to take indirect paths.

Travel times generally worsened as time went on, up until the stadium traffic ended. At this point, traffic dramatically speeds up by 25-45% as seen in Figure 7. As expected, the road with no road failures performed the best by 5-10%. Contributors to total travel time appear in Figure 8 as a surface, taken from Simulation 1. An earlier start time coupled with greater epiphany time produce the most favorable conditions. Consistent with Figure 7, a delayed start time degrades travel time regardless of the total length of a driver's epiphanies.

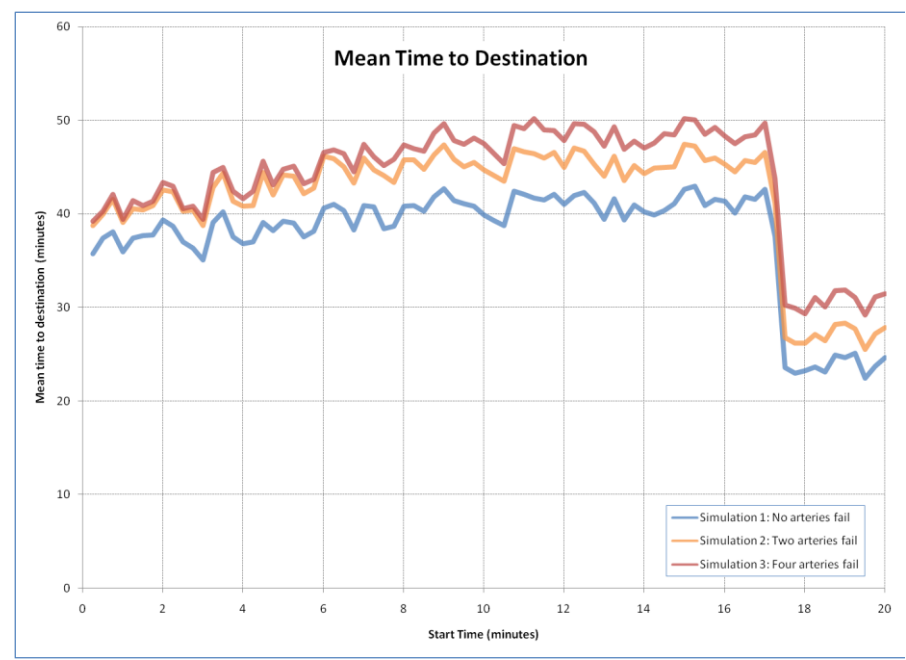


Figure 7: Mean time to destination

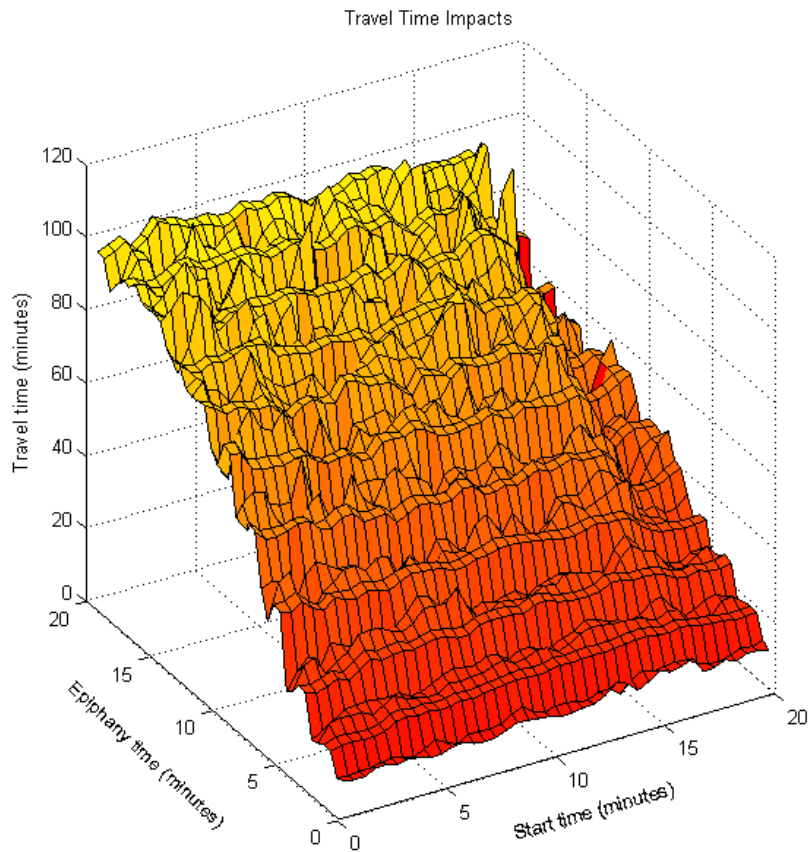


Figure 8: Travel time impacts

In all cases, Dijkstra's Algorithm performance was sufficient. To interject randomness, the normal, random layer and the epiphany driver implementation worked well. Each run took longer than expected, about 110 hours, with simulations modeling bad edges taking longer.

The summarized results are:

1. Dijkstra's algorithm was a sufficient approach to large-scale pathing problems. In all cases, it accurately routed vehicles to the best of its knowledge.
2. The algorithm as implemented was capable of successfully navigating all tested scenarios. This included grids with unreachable destinations, edges that sporadically failed and, when provided, accurate pathing adjustments in line with upcoming traffic conditions.
3. Dijkstra's algorithm is too accurate for such a random system. Other methods, such as random driver speeds, a unique speed map for each driver and giving a select few drivers knowledge of upcoming traffic conditions helped randomize pathing decisions.
4. Such a system modeling over 50,000 cars can calculate and dynamically update every single car's pathing until all vehicles reach their destinations in a reasonable period of under five days using a relatively high-level programming language (MATLAB).
5. A positive relationship exists between the amount of epiphany time given to a driver and the time it takes to navigate a grid.
6. When edges fail, there is a non-trivial time impact on the system as a whole as cars re-route to nearby unblocked streets and arteries. For the simulations that ran, the time penalty for four versus two failed arteries was noticeable but not significantly slower.
7. While a start, such a model can improve by adding signaling, lane awareness and other impacting variables.
8. While a start, such a model can improve by adding signaling, lane awareness and other impacting variables.

Formatted: Outline numbered + Level: 2 +
Numbering Style: 1, 2, 3, ... + Start at: 1 +
Alignment: Left + Aligned at: 0.25" + Indent
at: 0.55"

9. Arteries may not always be the point of congestion. As main roads are degraded in speed, side streets may experience as much or heavier congestion.

10.7. Works Cited

Abramowitz, M., & Stegun, I. A. (Eds.). (1972). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. New York: Dover Publications, Inc.

Bellomo, N., Delitala, M., Coscia, V., & Brezzi, F. (2002). On the Mathematical Theory of Vehicular Traffic Flow I: Fluid Dynamic and Kinetic Modelling. *Mathematical Models & Methods in Applied Sciences*, 1801-1843.

Cormen, T. H., Lieserson, C. E., Rivest, R. L., & Clifford, S. (2001). *Introduction to Algorithms*. Cambridge: MIT Press.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 269-271.

Doumith, E. A., Gagnaire, M., Audouin, O., & Puech, N. (2005). Traffic engineering for Virtual Private Networks and random traffic demands in WDM optical core networks. *14th International Conference on Computer Communications and Networks* (pp. 317-322). Tokyo: Japan.

Huang, B., Wu, Q., & Zhan, F. B. (2007). A shortest path algorithm with novel heuristics for dynamic transportation networks. *International Journal of Geospatial Information Science*, 625-644.

Hunter, J. S. (1986). The Exponentially Weighted Moving Average. *Journal of Quality Technology*, 18 (4), 203-207.

Leurent, F., & Aguilera, V. (2005). An Atomic Dijkstra Algorithm for Dynamic Shortest Paths in Traffic Assignment. *10th EWGT Meeting and 16th Mini-EURO Conference*. Poznan.

Luyao, C., Zhou, J., Li, J., & Chen, Y. (2007). Bidirectional Dijkstra algorithm for best-routing of urban traffic network. *Geoinformatics*, 6754-43.

Transportation Research Board of the National Academies. (2000). *Highway Capacity Manual 2000*. Washington: Transportation Research Board.

U.S. Census Bureau. (2006). *Selected Social Characteristics in the United States: 2006*. Retrieved 10 12, 2008, from http://factfinder.census.gov/servlet/ADPTable?_bm=y&-geo_id=16000US0827425&-qr_name=ACS_2006_EST_G00_DP2&-ds_name=ACS_2006_EST_G00_-&-_lang=en&-redoLog=false&-_sse=on

Zhang, F. B., & Noon, C. E. (1998). Shortest path algorithms: An evaluation using real road networks. *Transportation Science*, 65-73.

Zheng, J., & Mouftah, H. T. (2004). *Optical WDM Networks*. Hoboken: John Wiley & Sons, Inc.

Formatted: Outline numbered + Level: 1 + Numbering Style: 1, 2, 3, ... + Start at: 1 + Alignment: Left + Aligned at: 0" + Indent at: 0.25"

Formatted: No underline, Font color: Custom Color(RGB(54,95,145))

8. Appendix

1. Source Code

MATLAB source code and this article is available on the Internet at the non-profit Web site Archive.org.

Study (PDF): <http://www.archive.org/download/ScottFriend/TrafficStudy.pdf>

Study (DOCX): <http://www.archive.org/download/ScottFriend/TrafficStudy.docx>

Source tar archive: <http://www.archive.org/download/ScottFriend/TrafficStudyCode.tar>

Source zip archive: <http://www.archive.org/download/ScottFriend/TrafficStudyCode.zip>

Zone	Percent	Cars	Sum
1	2.00%	1,958	1,958
2	2.00%	1,958	3,916
3	0.50%	490	4,406
4	5.00%	4,895	9,301
5	4.00%	3,916	13,217
6	3.00%	2,937	16,154
7	0.50%	490	16,643
8	6.00%	5,874	22,517
9	6.00%	5,874	28,391
10	1.00%	979	29,370
11	6.00%	5,874	35,244
12	4.00%	3,916	39,160
13	7.00%	6,853	46,013
14	6.00%	5,874	51,887
15	5.00%	4,895	56,782
16	8.00%	7,832	64,614
17	8.00%	7,832	72,446
18	5.00%	4,895	77,341
Roamers	4.00%	3,916	81,257
Stadium	17.00%	16,643	97,900

Table 2: Vehicle departures by zone

ces
11.

Exit Artery	Car Usage
W. Mulberry St.	13,916
E. Prospect Rd.	20,465
E. Drake Rd.	22,419
E. Horsetooth Rd.	23,967
W. Harmony Rd.	13,217
Other	3,916

Table 3: Exit artery usage by car volume

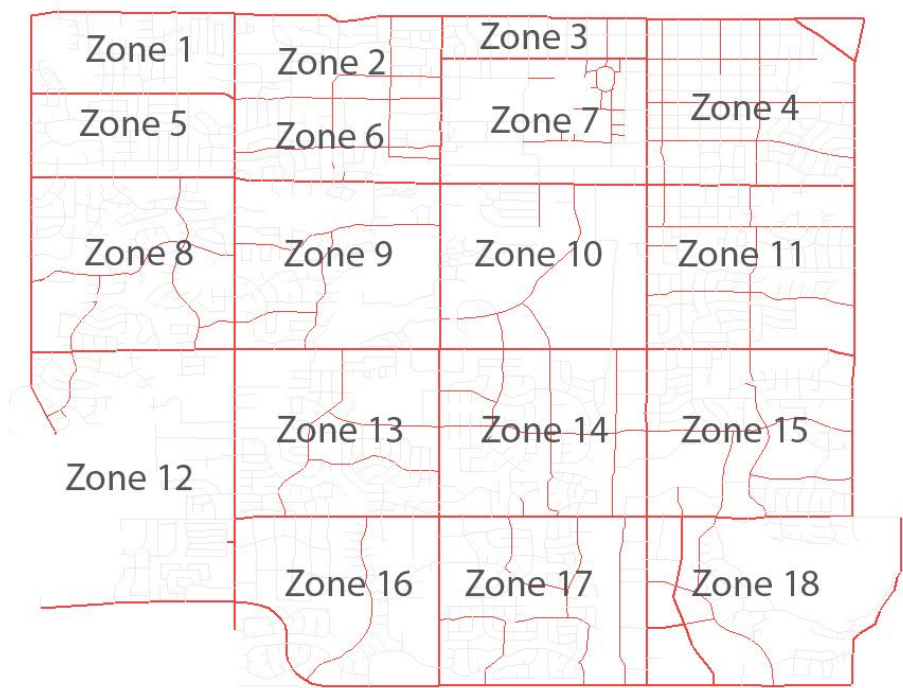


Figure 97: Map zones

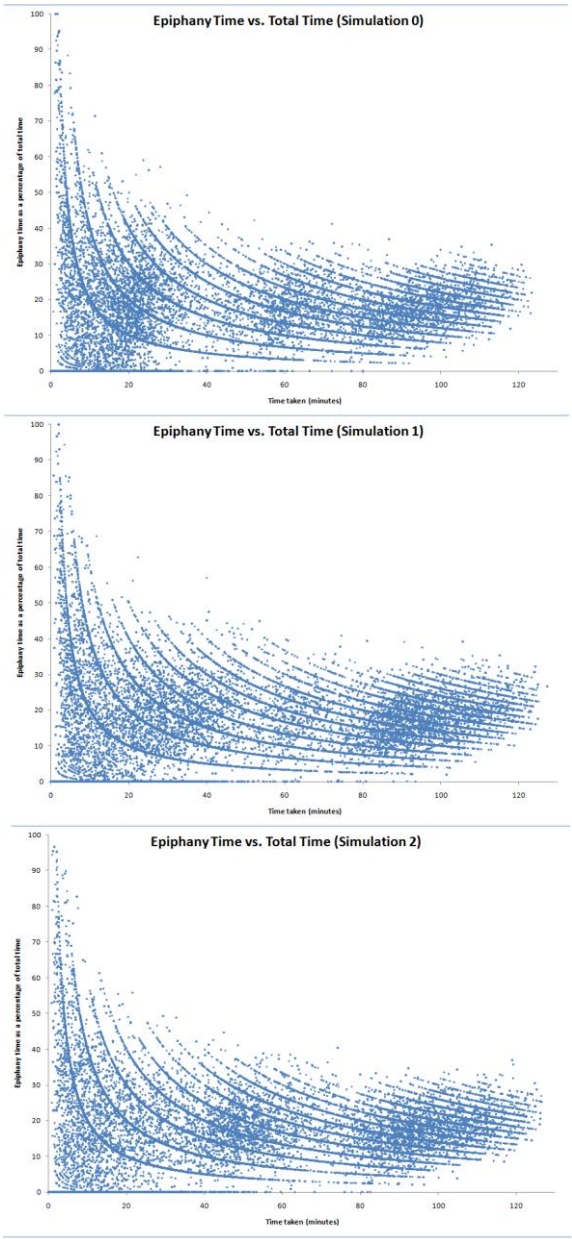


Figure 108: Epiphany time as a function of total time

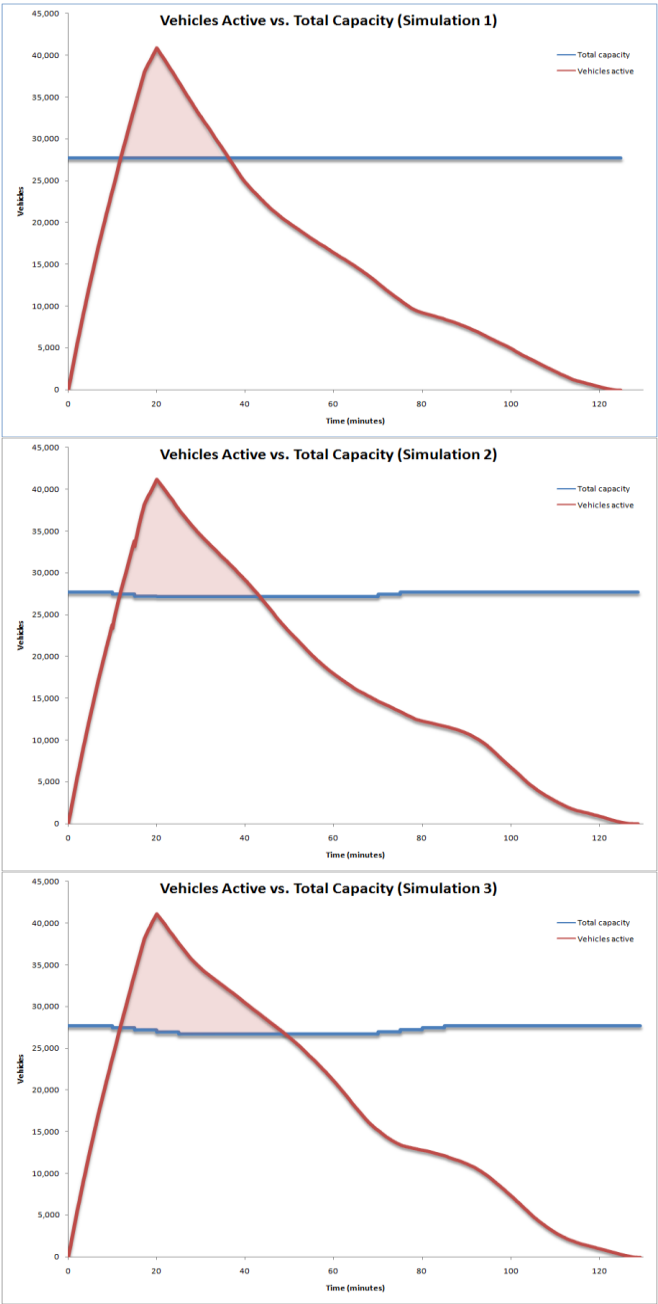


Figure 119: Vehicles active vs. total capacity

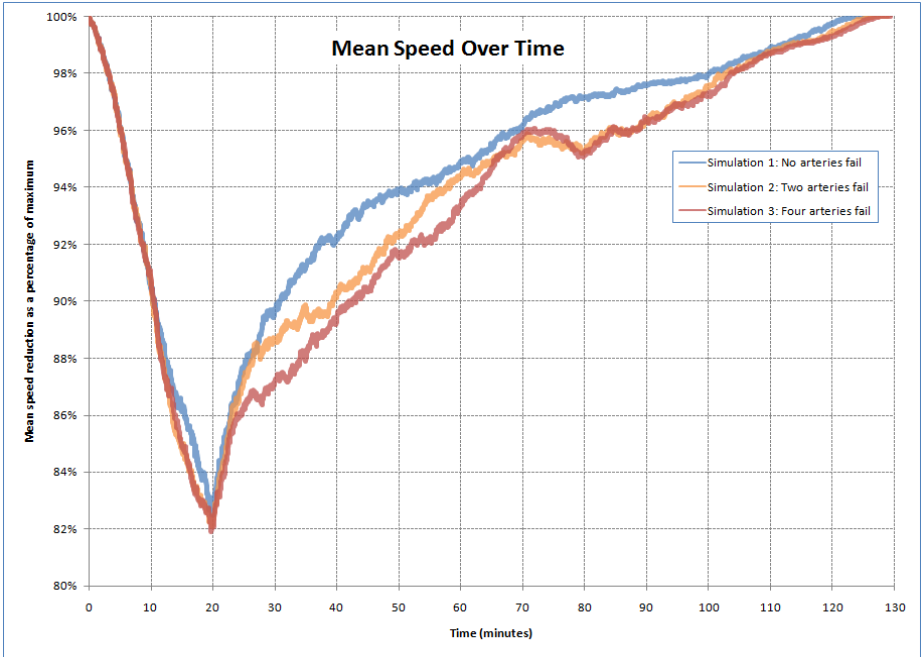


Figure 1210: Mean speed over time

Formatted: Left, Space After: 0 pt, Line spacing: single

Formatted: Space After: 0 pt, Line spacing: single

Formatted: Font: 11 pt, Not Highlight

Formatted: Normal, Left, Indent: Left: 0",
Line spacing: single